Navy Personnel Research and Development Center

San Diego, CA 92152-6800 TN-90-17 April 1990

The MS-DOS Routing and Configuration Program Design

Brian Van de Wetering
Brian Thomason

NPRDC-TN-90-17 April 1990

The MS-DOS Routing and Configuration Program Design

Brian Van de Wetering
Brian Thomason
Systems Engineering Associates
San Diego, California 92109

Reviewed and released by
Wallace H. Wulfeck II
Director, Training Technology Department

Navy Personnel Research and Development Center
San Diego, California 92152-6800

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATE COVERED
	April 1990	Interim--Sep 88-Feb 90

4. TITLE AND SUBTITLE
The MS-DOS Routing and Configuration Program Design

5. FUNDING NUMBERS
Program Element 0604722J
Work Unit 99-PJ1-90-006

6. AUTHOR(S)
Brian Van de Wetering and Brian Thomason

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Navy Personnel Research and Development Center
San Diego, California 92152-6800

8. PERFORMING ORGANIZATION REPORT NUMBER
NPRDC-TN-90-17

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of the Assistant Secretary of Defense
(Force Management and Personnel) (Room 3E808, Pentagon)
Washington, DC 20301-0000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Many vendors produce high-performance, low-cost training hardware, but bundle their products with proprietary software interfaces. Because these interfaces are proprietary, courseware and authoring systems written to operate on one set of hardware will not run on a competitor's hardware. Expensive reprogramming is needed to adapt to new hardware. These reprogramming costs can be eliminated by adopting standard software interfaces. The objectives of this effort were to describe and develop a standard software interface that will allow training systems to be assembled from separate "plug-and-play" components in the same way that stereo systems can be assembled from separate speakers, amplifiers, and other components. The Portable Courseware (PORTCO) architecture consists of two interfaces, the Device Services Interface and the Device Handler Interface. It also contains three layers: application, routing and configuration, and device handler layer. This architecture should allow applications software to run on any compliant set of hardware components. The series of reports describing the PORTCO architecture should direct development of portable MS-DOS applications and standard peripheral device handlers. This report is intended for system vendors, and describes the design of the first PORTCO routing and configuration program.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Courseware portability, computer-based training, interactive courseware, virtual device interface	32
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

Foreword

This document describes the design of the routing and configuration program, a component of an architecture for interactive courseware delivery systems. It was developed under the sponsorship of the Office of Secretary of Defense (Force Management and Personnel). Mr. Gary Boycan was the technical sponsor. Funding was provided under Program Element 0604722J, Work Unit 99-PJ1-90-006.

This document was developed under the technical supervision of Dr. Raye Newmen, Dr. Wallace H. Wulfeck, and Mr. Walter F. Thode of the Navy Personnel Research and Development Center (NPRDC). This report itself was written by Systems Engineering Associates under Contract N66001-88-D-0054, Delivery Order 7J30.

This architecture was developed as part of an effort to complete a reference implementation of a courseware portability specification. The implementation is scheduled for later this year. Comments on the architecture described here are solicited from all interested parties. The specification will then be submitted for consideration for official adoption by the Department of Defense and the National Institute of Standards and Technology.

Point of contact at NPRDC is Walter F. Thode (619) 553-7703 or AUTOVON 553-7703.

WALLACE H. WULFECK II
Director, Training Technology Department

Summary

Background

Over the next several years the Federal Government will invest millions of dollars to develop training materials for delivery on computer-based interactive training systems. To support this investment, Federal agencies will acquire a variety of computers and peripheral devices. This hardware will host several operating systems and authoring system software to speed courseware development.

Many vendors produce high-performance, low-cost training hardware, but bundle their products with proprietary software interfaces. Because these interfaces are proprietary, courseware and authoring systems written to operate on one set of hardware will not run on a competitor's hardware. Expensive reprogramming is needed to adapt to new hardware. These reprogramming costs can be eliminated by adopting standard software interfaces.

Objectives

The objectives of this effort were to describe and develop a standard software interface that will allow training systems to be assembled from separate "plug-and-play" components in the same way that stereo systems can be assembled from separate speakers, amplifiers, and other components.

Approach

The Portable Courseware (PORTCO) architecture consists of two interfaces, the Device Services Interface and the Device Handler Interface. It also contains three layers: application, routing and configuration, and the device handler. This architecture should allow applications software to run on any compliant set of hardware components.

Results and Conclusions

This report is the fourth in a series of five reports; it is intended primarily for system vendors and describes the design of the first PORTCO routing and configuration program. The first report provides an overview of the PORTCO architecture and should be of interest to all who are concerned with computer-based training. The second report describes the MS-DOS Device Services Interface and is intended primarily for programmers who want to develop portable application software. The third report describes the Device Handler Interface and should be of primary interest to device manufacturers

and system vendors who must develop device handler software. The fifth report is intended as additional support for those who must develop complaint MS-DOS device handlers.

Recommendations

1. The reports describing the PORTCO architecture should direct development of portable MS-DOS applications and standard peripheral device handlers.

2. The architecture should serve as a foundation for compliance with the Interactive Video Industry Association's "Recommended Practices for Interactive Video Portability." The architecture should motivate development of specific applications and device handlers that adhere to this specification.

3. Feedback about problems and suggested improvements should be forwarded to the Navy Personnel Research and Development Center, Code 152.

Table of Contents

List of Tables

List of Figures

1. Purpose and Scope

Over the next several years the Federal Government will invest millions of dollars to develop training materials for delivery on computer-based interactive training systems. Several commercial manufacturers currently offer high-performance, low-cost interactive training hardware but bundle their products with proprietary software interfaces. Because these interfaces are proprietary, courseware and authoring software written to operate on one product will not run on a competitor's equipment without expensive reprogramming.

These reprogramming costs can be eliminated by adopting standard software interfaces to shield courseware and authoring software from changes in training system hardware. Two such interfaces and a portable courseware (PORTCO) architecture are specified in the publications listed in Section 9. Figure 1 illustrates the PORTCO architecture and its two standard interfaces, the Device Services Interface (DSI) and the Device Handler Interface (DHI).

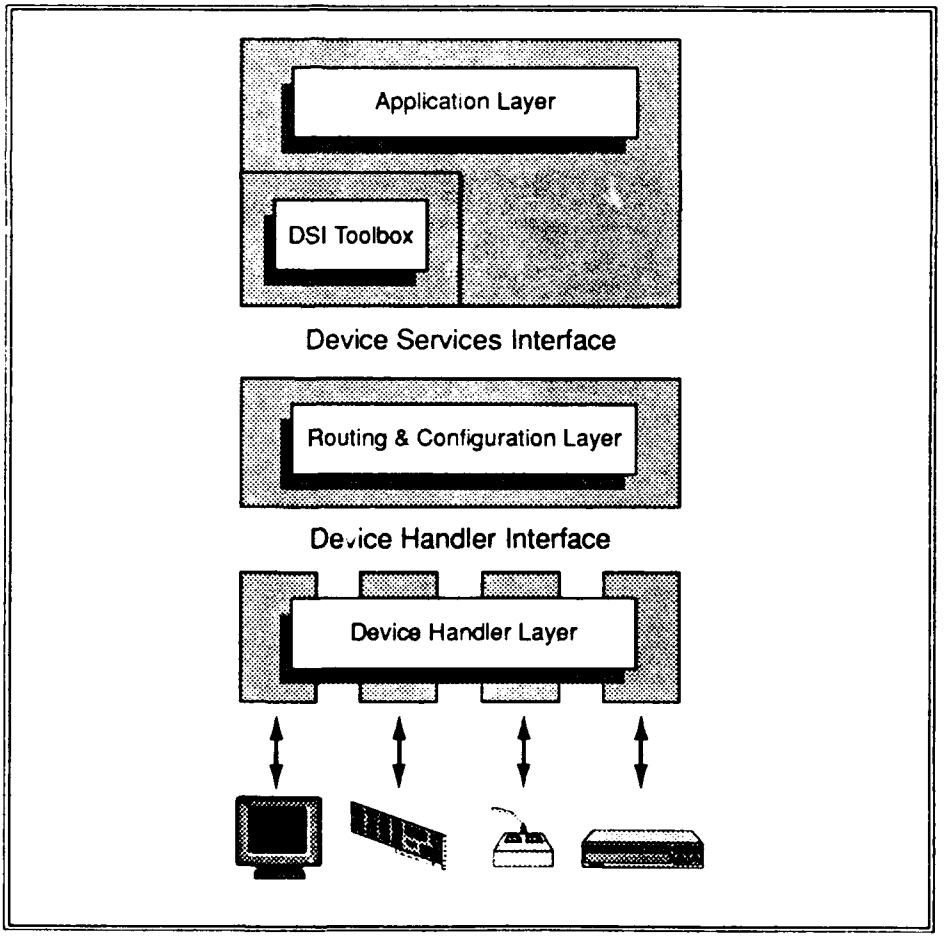


Figure 1
The PORTCO architecture contains three layers and two interfaces.

This publication presents a design for a program (named the R&C program) that implements Figure 1's routing and configuration (R&C) layer on MS-DOS computers. It is intended to guide coding of this

program by employees of Systems Engineering Associates, and to serve as an example for other programmers developing PORTCO-compliant R&C-layer software. The design presented here is one of many designs that might be used to implement this layer; it is not a specification for all future R&C programs.

This is one of several documents, listed in Table 1, describing the PORTCO architecture in an MS-DOS environment. All readers should be familiar with the first three documents in Table 1 and should be accomplished MS-DOS system programmers with a sound understanding of "C" and assembly language.

Table 1
This specification is one of a collection of PORTCO documents.

Title	Description
A Portable Courseware Architecture	A top-level description of the PORTCO architecture.
The MS-DOS Device Services Interface	A specification for the MS-DOS interface between the PORTCO architecture's Application layer and its R&C layer. This specification guides the development of compliant applications and compliant R&C programs.
The MS-DOS Device Handler Interface	A specification for the MS-DOS interface between the PORTCO architecture's R&C layer and its Device Handler layer. This specification guides development of compliant MS-DOS device handlers and compliant MS-DOS R&C programs.
MS-DOS Routing and Configuration Program Design	A specification for the first implementation of compliant MS-DOS R&C-layer software, expected to serve as an example for future implementations.
Guidelines for Implementing MS-DOS Device Handlers	A collection of informal guidelines and examples to support the development of compliant MS-DOS device handlers.

Section 2 presents a summary of the R&C program's requirements, collected from Table 1's first three documents. Section 3 provides an overview of the R&C program's design, and Sections 4 through 7 provide detailed descriptions of the program's code partitions and data structures. Section 8 presents the program's error messages, and Section 9 is a list of references.

2. Requirements Summary

The R&C program implements the PORTCO architecture's R&C layer. Because it provides DSI services to applications using the DHI, the R&C program must adhere to the conventions of both the DSI and the DHI. References Thomason, Van de Wetering and Booth (1990), Thomason and Van de Wetering (1990) and Van de Wetering and Thomason (1990) describe these interfaces in detail. The following list summarizes the R&C program's requirements and references other PORTCO publications that contain more complete descriptions of these requirements.

- The R&C program must read the DSI configuration file. Section 4 of Thomason and Van de Wetering (1990) describes this file's format and contents.
- The R&C program must load and initialize device handlers for the system configuration specified in the DSI configuration file. Section 3 of Van de Wetering and Thomason (1990) describes loading device handlers. Section 4 of Van de Wetering and Thomason (1990) describes device handler initialization.
- The R&C program must examine an identification stamp near each device handler's load address to verify that a valid handler has been loaded. Section 3 of Van de Wetering and Thomason (1990) describes the device handler identification stamp.
- The R&C program must map each logical device to a device handler entry point. Section 3 of Thomason, Van de Wetering and Booth (1990) describes the mapping of logical devices to device handlers.
- The R&C program must load a software interrupt vector with the entry point of a routine to route packets to device handlers. Section 4.1 of Thomason and Van de Wetering (1990) describes this interrupt and its specification in the DSI configuration file.
- The R&C program must return control to MS-DOS leaving its interrupt service routine and device handlers in memory. Section 3 of Thomason and Van de Wetering (1990) describes the initialization of the DSI by the R&C program.
- The R&C program must pass packets from applications to device handlers and from device handlers to applications. Section 5.1 of Thomason and Van de Wetering (1990) and Van de Wetering and Thomason (1990) describe packet passing conventions.
- The R&C program must provide the R&C services described in Appendix A of Thomason and Van de Wetering (1990).

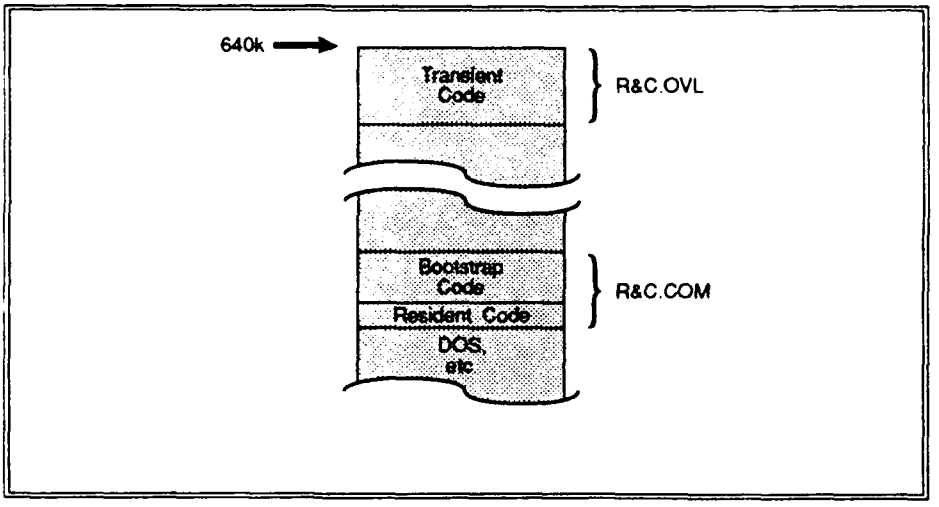
3. Design Overview

The R&C program is divided into three parts: bootstrap code, transient code, and resident code. Bootstrap code loads and executes transient code. Transient code loads and initializes device handlers. Resident code routes packets and performs the DSI's R&C services.

These three parts are implemented as two executable files, named R&C.COM and R&C.OVL. R&C.COM contains resident and bootstrap code and R&C.OVL contains transient code.

The R&C program is invoked by executing R&C.COM from the MS-DOS command line. Resident code in this file is loaded as low as possible in memory. Bootstrap code is loaded above resident code. Transient code in R&C.OVL is loaded as high as possible in memory to leave room for device drivers. Figure 2 illustrates the memory arrangement of bootstrap, transient, and resident code.

Figure 2
The transient code must be loaded as high as possible in memory.



Memory occupied by bootstrap code is reused as a resident code data area. Memory occupied by transient code is returned to the operating system when all device handlers are loaded and initialized. Resident code remains in memory until the DSI is terminated or the system is rebooted.

The following events occur when R&C.COM is executed.

- DOS loads the R&C program into memory at the current transient program area (TPA) address.
- The bootstrap code loads the transient code into the highest possible memory location.
- The bootstrap code pushes the addresses of resident code routines and data structures onto its stack for use by the transient code.

- The bootstrap code jumps to the transient code.
- The transient code parses the configuration file.
- The transient code initializes a jump table that will contain the entry point addresses of all the device handlers. Section 4 describes this jump table.
- The transient code initializes a table that contains the stack pointers (SS:SP) for all the device handlers.
- The transient code loads and initializes each device handler in the configuration file.
- The transient code puts each device handler's entry point address into the jump table.
- The transient code puts each device handler's initial stack pointer (SS:SP) into the stack table.
- The transient code saves the request and alert interrupt vectors.
- The transient code sets the request interrupt vector using the address passed by the bootstrap code.
- If this processing is successful, the transient code issues a terminate-but-stay-resident call to MS-DOS, leaving the resident code and the device handlers in memory.
- If this processing is unsuccessful, the transient code displays an error message on the system console and exits to MS-DOS, leaving nothing in memory.

4. Device Handler Jump Table

The device handler jump table (DHJT) is a dynamically allocated one-dimensional array of 32-bit addresses used by the R&C program to store the entry points of all the device handlers it loads. The device class and device number to which a device handler has been assigned determine the index in the DHJT where its entry point is stored.

The number of entries in the DHJT is determined by the device classes and device numbers referenced in the DSI configuration file. The R&C program's transient code must determine the size of the DHJT by reading the configuration file, then allocate the memory for the DHJT before loading any of the device handlers.

The DHJT is partitioned by device class into blocks of contiguous entries. The DHJT contains a partition for each device class referenced in the configuration file, a partition for R&C services, plus an extra partition for all device classes not referenced in the configuration file.

Figure 3 illustrates DHJT partitioning for a sample configuration file. This configuration file references three device classes (overlay, xy, and videodisc). The corresponding DHJT therefore contains five partitions: one for each class referenced in the configuration file, one for R&C services, and one for all classes not mentioned in this file.

Each partition in the DHJT contains the same number of entries: one entry for each device number referenced in the configuration file, one entry for device number 255 (R&C services), and one entry for all device numbers not referenced in this file.

For example, if two device numbers appear in the DSI configuration file (e.g., 0 and 1), each DHJT partition will contain four entries: one for device number 0, one for device number 1, one for device number 255, and one for all other device numbers. Figure 4 illustrates the assignment of DHJT entries for such a configuration file. Device numbers are printed in boldface in this figure's sample configuration file for emphasis.

It is important to note that the size of each DHJT partition is governed by how many device numbers are referenced in the DSI configuration file, not by the range of numbers referenced in the file. For example, if device numbers 0 and 3 appear in the configuration file, each DHJT partition will still contain only four entries: one for device number 0, one for device number 3, one for device number 255, and one for all other device numbers. Figure 5 illustrates the assignment of DHJT entries for such a file.

Figure 3
The DHJT is partitioned by device class.

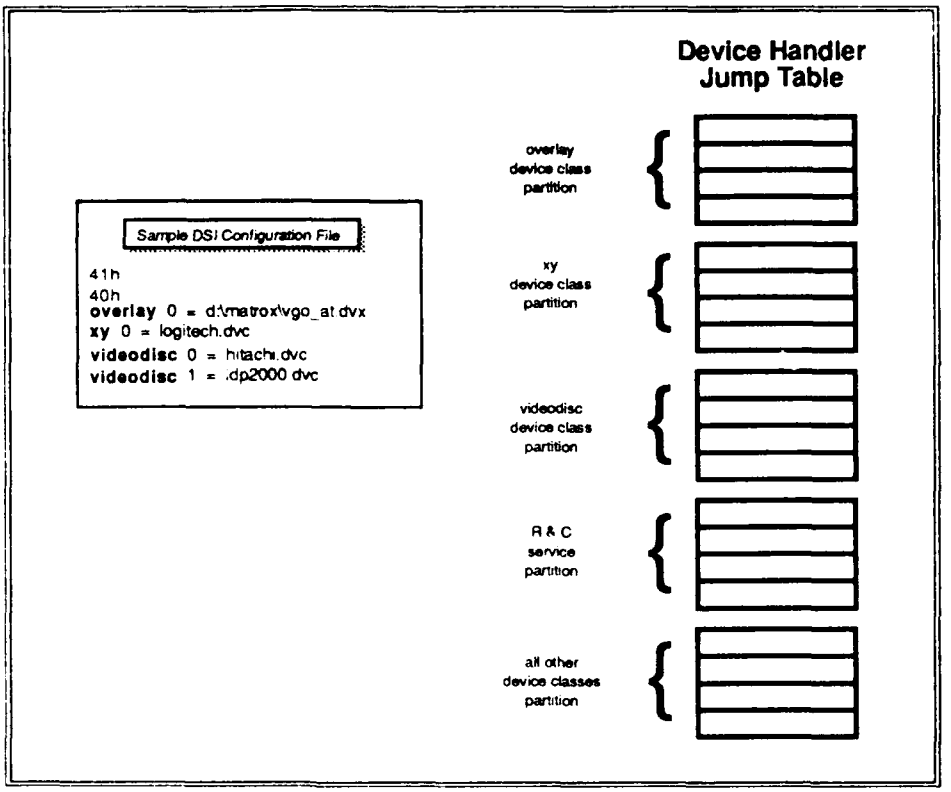
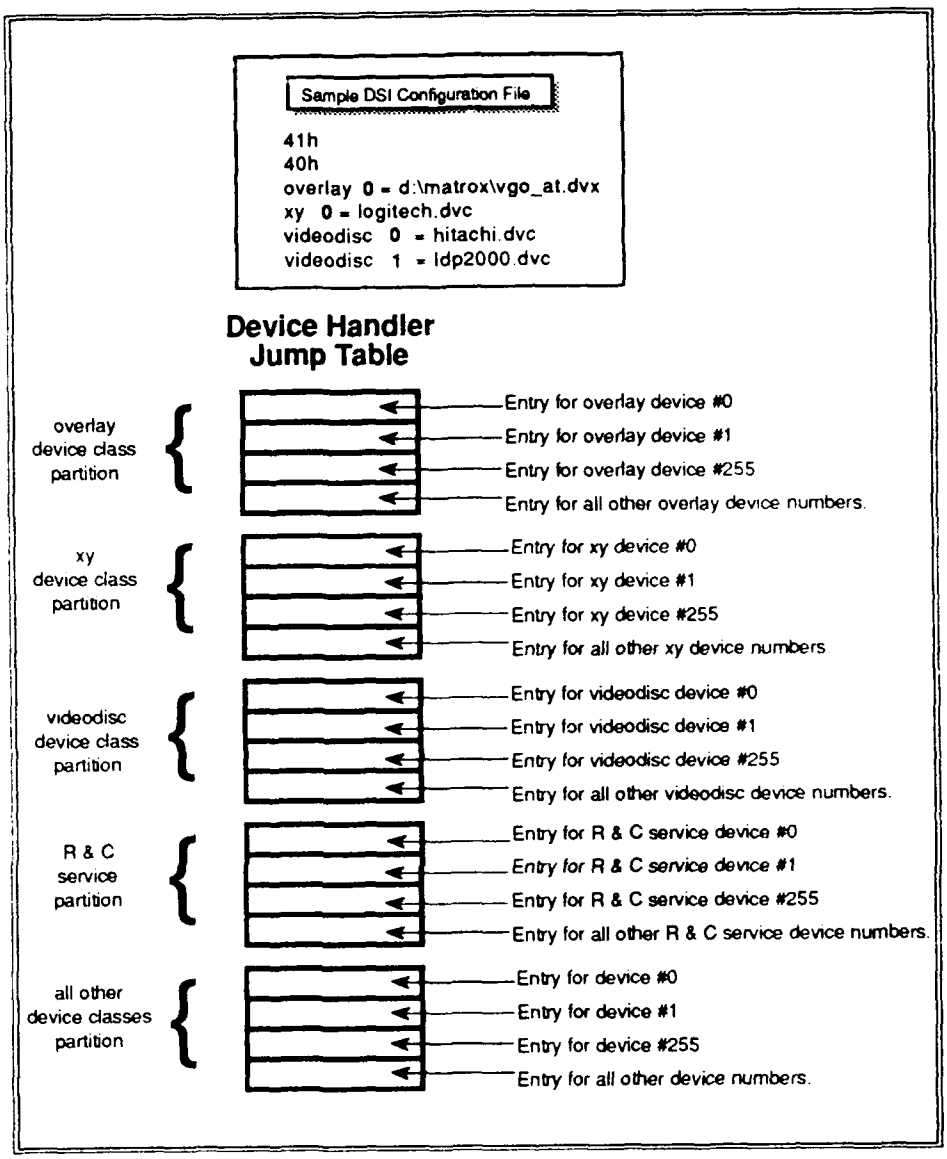


Figure 4
DHJT entries in each partition correspond to device numbers.



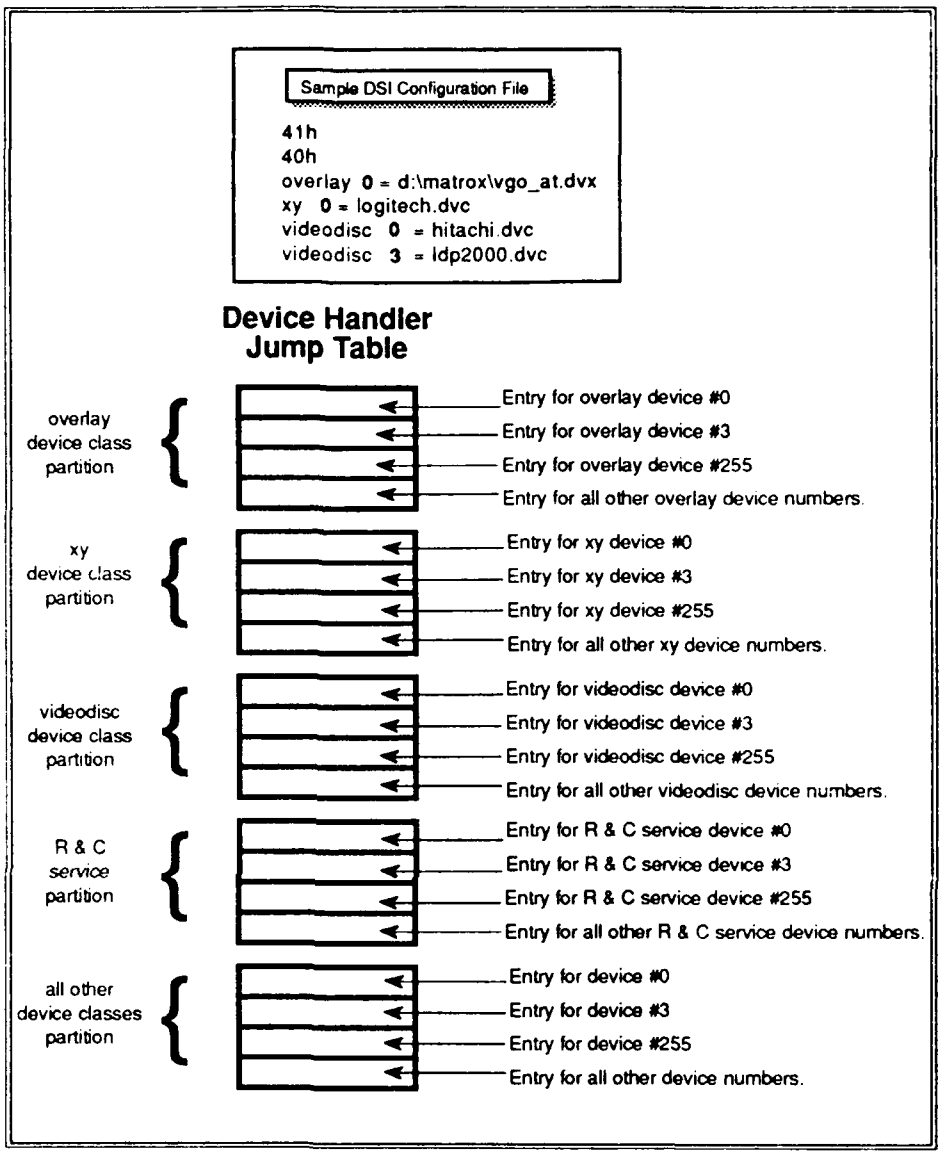


Figure 5
The size of each DHJT partition is governed by how many device numbers are referenced in the DSI configuration file.

The total number of entries in the DHJT is computed with Formula 1.

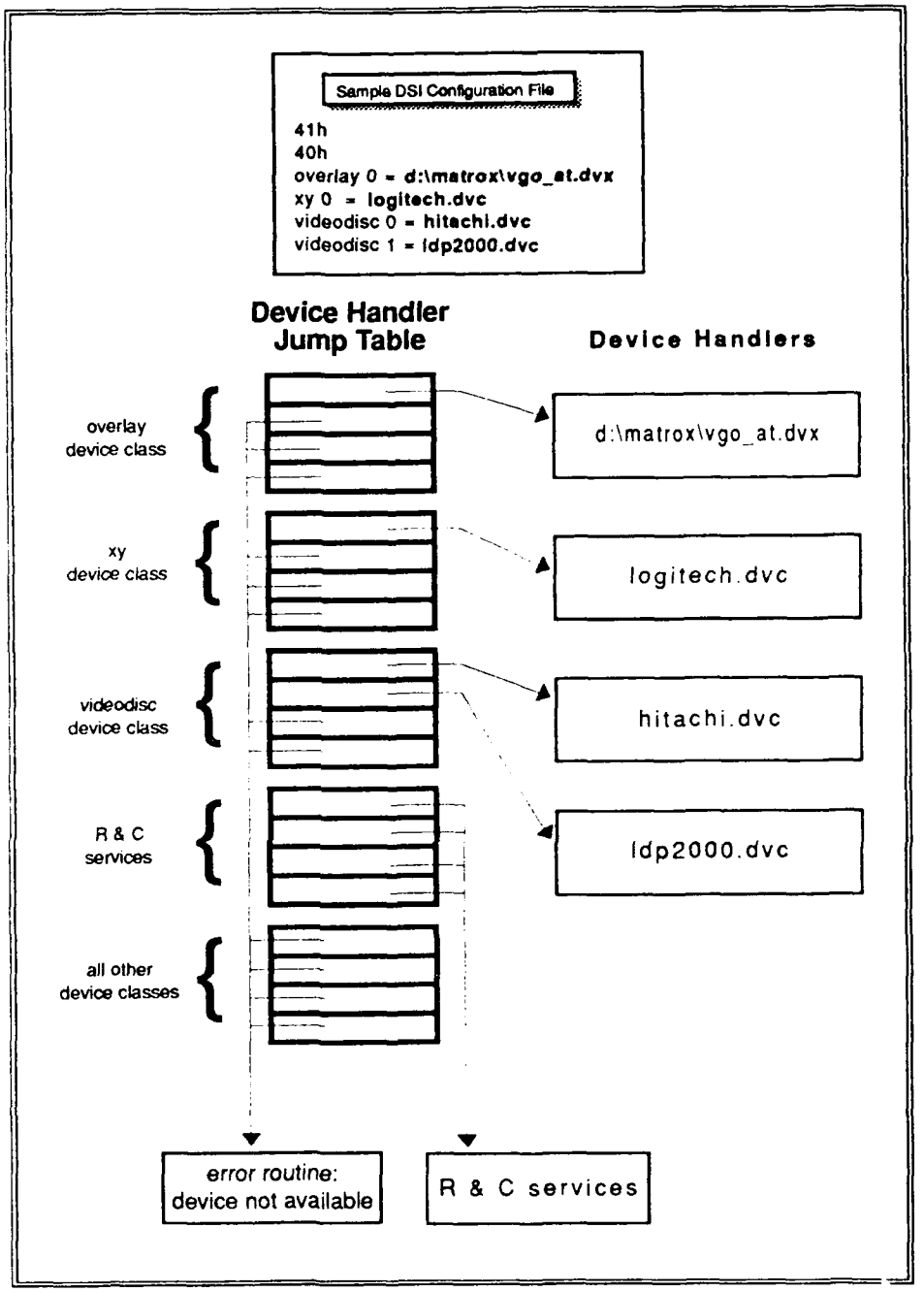
Formula 1
Number of entries in the DHJT.

(# of device classes referenced + 2) x (# of device numbers referenced + 2)

Because each entry is 32 bits long, the total number of bytes needed for the DHJT is four times the number of entries.

Each DHJT entry contains the entry point address of a device handler, of a routine that performs R&C services, or of an R&C program error routine that reports to the application that the requested device is not available. Figure 6 illustrates how entries in the DHJT are filled in with these entry point addresses. Note that all entries in the DHJT that have been assigned to a device class and number referenced in the configuration file contain the entry point address of a device handler.

Figure 6
DHJT entries contain device handler entry points or an error routine entry point.



To accomplish the mapping between a logical device and a DHJT index, the R&C program uses two tables: the device class look-up table (DCLT) and the device number look-up table (DNLT). Each of these tables occupies 256 bytes. The DCLT contains the index in the DHJT of the first entry in the DHJT partition for each device class. The DNLT contains the offset from the beginning of a DHJT partition to the DHJT entry for each device number. The DHJT index for a particular logical device is calculated by Formula 2.

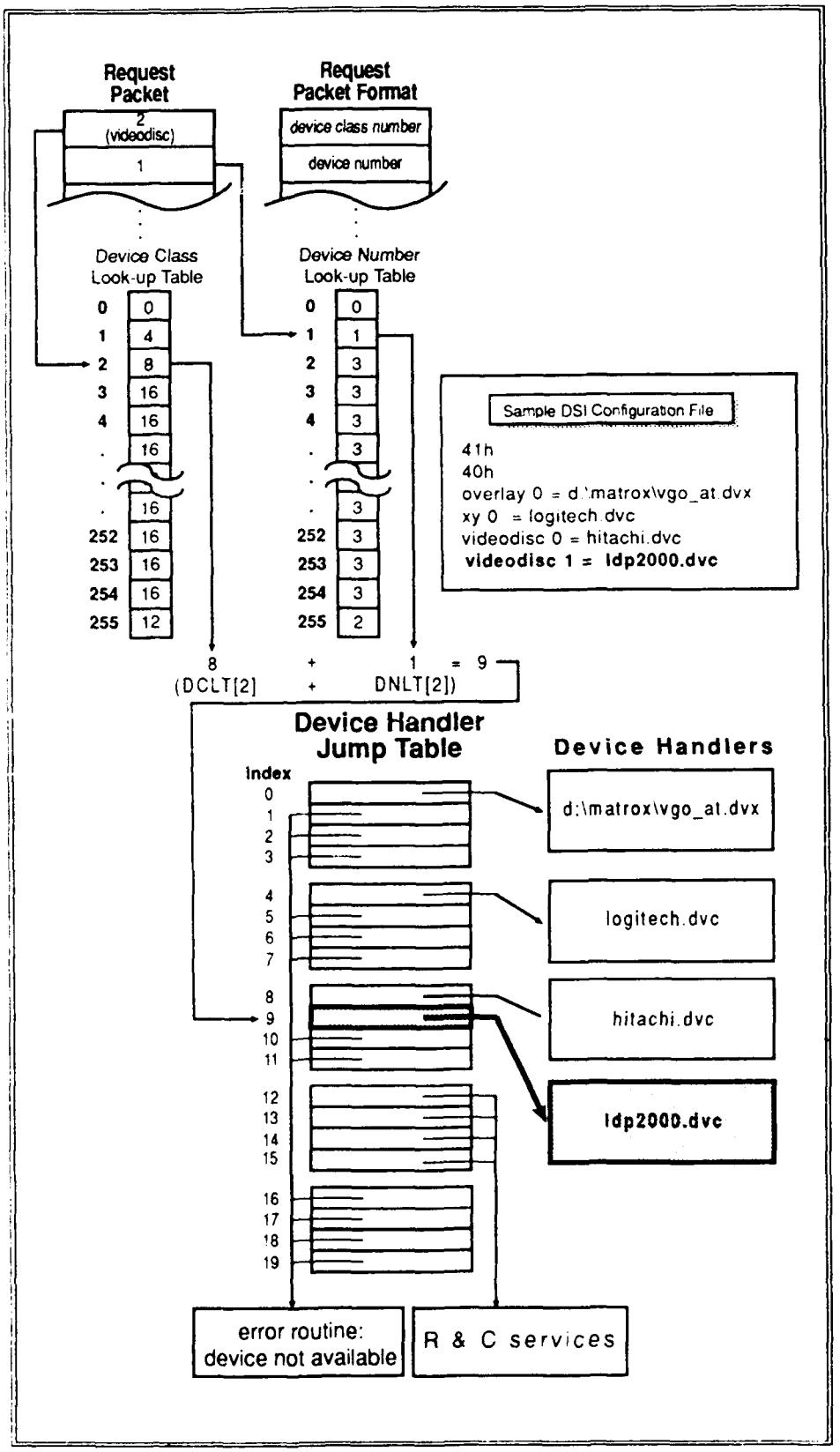
Formula 2
DHJT index for a logical device.

DCLT[device class number] + DNLT[device number]

Figure 7 shows an example of how the DCLT and DNLT are filled in and also illustrates how a DHJT index is calculated using Formula 2.

To pass a service request for a logical device to the appropriate device handler, the R&C program calculates the jump table index from the device class number and device number (in the request packet) using Formula 2, then issues a far call to the location stored at the calculated position in the DHJT.

Figure 7
The R&C program calculates a DHJT index using the DCLT and DNLT.



5. Stack Switching

The R&C program is responsible for switching stacks between the application and device handlers. Section 5.1 in Thomason and Van de Wetering (1990) and Van de Wetering and Thomason (1990) describe calling conventions between the application and the R&C program and between the R&C program and device handlers. To switch stacks, the R&C program uses a table of top-of-stack pointers (SS:SP) and a global variable to indicate which stack is currently active. This stack table contains an entry for each device handler and is indexed using the DCLT and DNLT in the same way as the DHJT. The R&C program also uses a variable to store the application's top of stack. Entries in the stack table are initialized by the transient code using the initial stack pointer extracted from each handler's id stamp.

To route a request, the R&C program's request ISR performs the following steps when invoked by an application. Steps shown in boldface are critical sections and must be performed with interrupts disabled. Pseudocode is included in parenthesis.

1. Calculates the handler's DHJT/stack table index using the DCLT and DNLT
   (DCLT[device class number] + DNLT[device number]).
2. **Saves the application's stack**
   (application_stack = SS:SP).
3. **Restores the handler's stack from the stack table**
   (SS:SP = stack_table[handler_index]).
4. **Records which stack is currently active**
   (current_stack = handler_index).
5. Calls the handler (call DHJT[handler_index]).
6. **Saves the value of the handler's stack upon return**
   (stack_table[current_stack] = SS:SP).
7. **Restores the application's stack**
   (SS:SP = application_stack).
8. **Records which stack is currently active**
   (current_stack = APPLICATION).

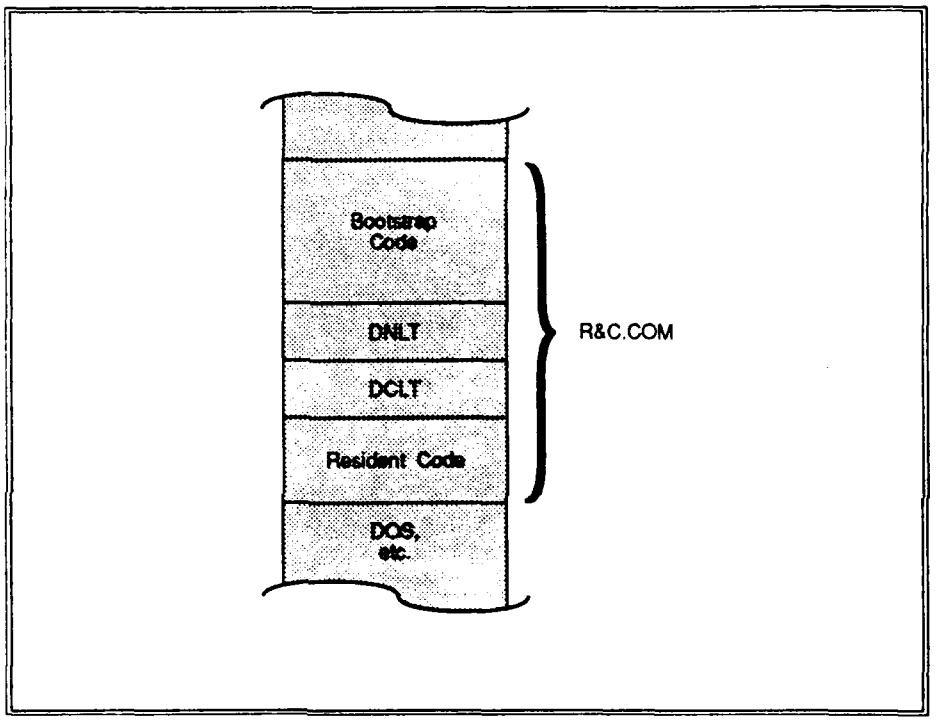
To route an alert, the R&C program performs the following steps when called by a device handler. The device handler does its alert processing using whatever stack is active when the hardware interrupt occurs. Steps shown in boldface are critical sections and must be performed with interrupts disabled.

1. Records which stack to restore on return
   (return_stack = current_stack).
2. If the current stack is not the application's stack
   (if (current_stack != APPLICATION)):
   a. **Saves the current stack**
      (stack_table[current_stack] = SS:SP).
   b. **Restores the application's stack**
      (SS:SP = application_stack).
   c. **Records which stack is currently active**
      (current_stack = APPLICATION).
   d. Records which stack to return to by pushing it onto the
      application's stack. This information is stored on the stack so
      that alerts can be nested
      (push return_stack).
3. Issues the alert interrupt.
4. Determines which stack to return to
   (pop return_stack).
5. **If the return stack is not the application's stack**
   (if (return_stack != APPLICATION)).
   a. **Saves the application's stack upon return**
      (application_stack = SS:SP).
   b. **Restores the previous stack**
      (SS:SP = stack_table[return_stack]).
   c. **Records which stack is currently active**
      (current_stack = return_stack).

6. Resident and Bootstrap Code

The R&C program's resident and bootstrap code, DCLT and DNLT are contained in the same executable file. Figure 8 illustrates the arrangement in memory of this file's code and data. Bootstrap code is highest in memory since it will be overwritten with device handlers. The jump table is second highest since its size is not known until the transient code has parsed the configuration file. Other code and data are below the jump table.

Figure 8
Bootstrap code is highest in memory since it will be overwritten with device handlers.



The bootstrap code loads and executes transient code. Transient code is loaded as high as possible in memory to leave room for device handlers. To do this, bootstrap code first computes a proposed load address. This address is as high in memory as the size of the transient code allows. Then the bootstrap code checks to be sure that the proposed load address is not in or below the bootstrap code itself. Next, the bootstrap code loads the transient code from disk using the DOS load overlay function (int 21, function 4Bh, al = 3). The bootstrap code then pushes the following addresses onto its stack for the transient code to use:

1. The entry point address of the resident code's interrupt service routine (ISR) that routes requests.
2. The entry point address of the resident code's routine that routes alerts.
3. The entry point address of the resident code's error routine that reports when a logical device is not available.

4. The entry point address of the resident code's routine that processes R&C services.
5. The address of the DCLT.
6. The address of the DNLT.
7. The starting address of the DHJT.
8. The address of a data area to store the previous contents of the request interrupt.
9. The address of the alert interrupt instruction to be patched.
10. The address of a data area in which to store the address of the stack table.

Finally, the bootstrap code jumps to the transient code. The transient code never returns control, but terminates leaving the resident code and device handlers in memory.

7. Transient Code

The transient code loads and initializes the device handlers specified in the configuration file, completes the DCLT, DNLT, and DHJT, and sets the request interrupt.

Because the transient code is loaded as an overlay and called by the bootstrap code, it behaves slightly differently from standard MS-DOS executables. The transient code has no Program Segment Prefix (PSP) of its own, so it must not try to access one. If it uses the bootstrap code's PSP, it must take into account that it may not have been loaded immediately adjacent to this PSP. Also, the transient code must move its parameters off the bootstrap code's stack before setting up its own stack. In other ways, the transient code is like any other MS-DOS executable.

The transient code performs the following actions:

- It sets up its data area.
- It moves parameters from the bootstrap code's stack and places them into its own data area.
- It sets up its own stack area and does other start-up tasks that stand-alone programs need to do.
- It parses the configuration file and builds a data structure using the file's contents.
- It computes the worst-case size of the DHJT and the stack table, assuming that all device handlers referenced in the configuration file will load and initialize successfully, and determines the load address of the first device handler based on this size. Figure 9 shows the location of the DHJT and load address of the first device handler.
- Based on the size of the DHJT, it computes the starting address of the stack table and stores it in the data area passed by the bootstrap code.
- It initializes all DHJT entries with the entry point address of the resident code's error routine that reports when a logical device is not available.
- It does the following for each device handler referenced in the configuration file:
  - Computes how much memory is still available.
  - Loads the device handler into memory using the DOS load overlay function (int 21, function 4Bh, al = 3).
  - Checks to see that the device handler's id stamp is present.

- Extracts the handler's initial SS:SP and puts it in the stack table.
- Calculates the handler's entry point.
- Issues DHI service 0 "Initialize Device Handler" using the initialization string from the configuration file.
- Puts the device handler's entry point address into the DHJT.
- Extracts the next device handler's load address from the packet returned from service 0.

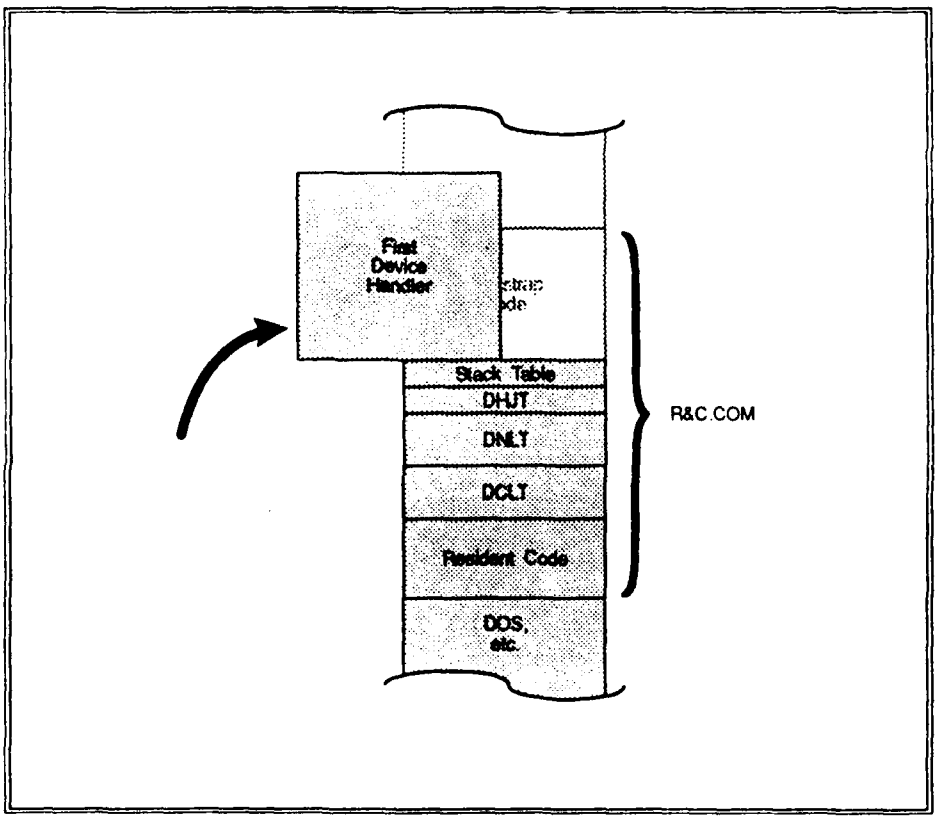


Figure 9
The transient code loads the first device handler immediately above the DHJT.

- It saves the original contents of the request interrupt vector in the data area passed by the bootstrap code.
- It resets the request interrupt vector with the entry point address of the routine in the resident code that routes requests. This address was passed on the stack by the bootstrap code.
- It patches the alert interrupt instruction with the value of the alert interrupt. The location of this instruction was passed on the stack by the bootstrap code.

- If no fatal errors were detected in the foregoing steps, it exits to MS-DOS using a TSR (int 21, function 31h) call. It passes to MS-DOS the number of memory paragraphs to leave resident. This number is the difference between the segment where the resident code was loaded and the segment returned by the last device handler loaded. Figure 10 shows the section of memory that will remain resident after the TSR call.

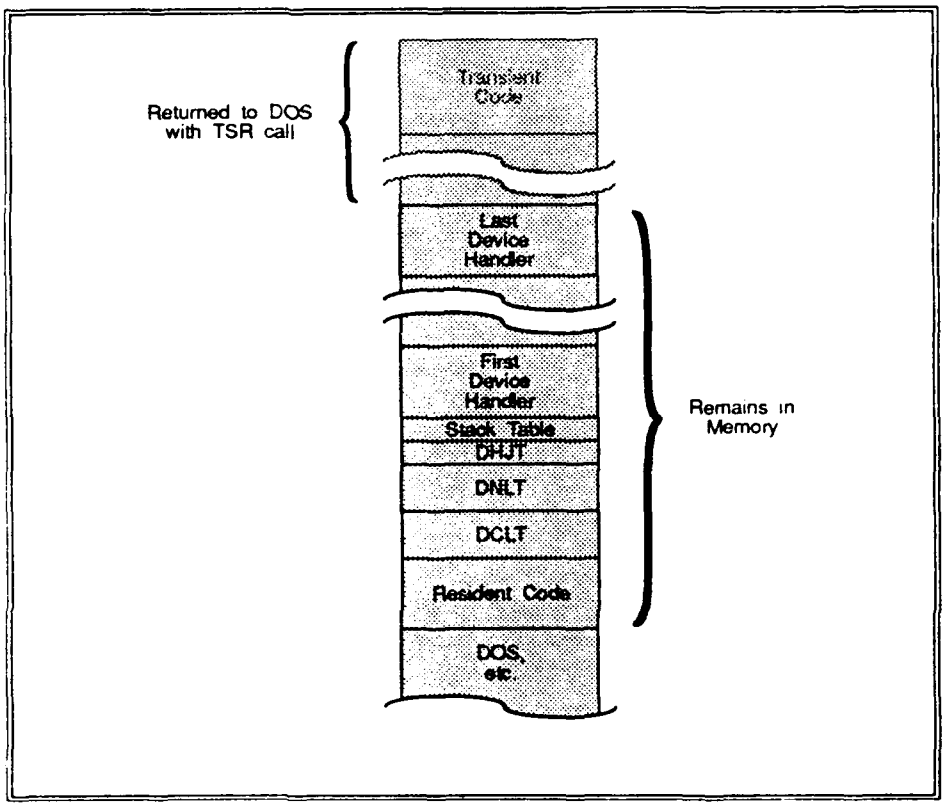


Figure 10
The resident code and the device handlers are left in memory with a TSR call.

- If a fatal error was detected at any point in the foregoing steps, the transient code displays an error message on the system console and exits to MS-DOS without staying resident. Section 8 lists all fatal error conditions and their corresponding error messages.

8. Error Handling

The R&C program reports two kinds of errors: fatal errors and recoverable errors. Fatal errors are those conditions that make it impossible for the R&C program to complete its job. Recoverable errors are those conditions from which the R&C program can recover and continue, but with less than expected functionality.

During initialization, when the R&C program encounters a fatal error it displays an error message and exits. When the R&C program encounters a recoverable error during initialization, it displays an error message, waits for a user acknowledgement, then performs any necessary recovery actions and continues. The R&C program uses the DOS display character function (int 21h, function 2) to display error messages. The R&C program uses the DOS keyboard input without echo function (int 21h, function 8) to get an acknowledgement from the user.

Tables 2 and 3 list the possible R&C program errors. In the error messages shown, words surrounded by brackets ("[]") should be replaced with the appropriate information at run-time.

Table 2
The R&C program reports three fatal errors.

Type of Error & Error Message
Configuration file missing: FATAL ERROR: Cannot find configuration file "CONFIG.DSI."
Syntax error: FATAL ERROR: Syntax error in configuration file "CONFIG.DSI" at line [N]
Insufficient memory: FATAL ERROR: Insufficient memory to continue

Table 3
The R&C program reports nine recoverable errors

Type of Error & Error Message	Recovery Action
Interrupt out of range: WARNING: Interrupt out of range in the configuration file "CONFIG.DSI" at line [N], using default values. Press any key to continue.	Use default values. The service interrupt defaults to 40h; the event interrupt to 41h.
Service interrupt same as event interrupt: WARNING: Service interrupt and event interrupt are the same, using default values. Press any key to continue.	Use default values. The service interrupt defaults to 40h; the event interrupt to 41h.
Unrecognized device class id: WARNING: Unrecognized device class id "[bad string]" in the configuration file "CONFIG.DSI" at line [N]. Skipping to next device. Press any key to continue.	Skip to next device in configuration file.
Device number out of range: WARNING: Device number out of range in the configuration file "CONFIG.DSI" at line [N]. Skipping to next device. Press any key to continue.	Skip to next device in configuration file.
Duplicate device number in device class: WARNING: Duplicate device number [N] in device class "[device class string]" in the configuration file "CONFIG.DSI" at line [N]. Skipping to next device. Press any key to continue.	Skip to next device in configuration file.
Cannot find device handler: WARNING: Cannot find device handler "[device handler filename]". Skipping to next device. Press any key to continue.	Skip to next device in configuration file.
Device handler reports fatal error during initialization: WARNING: Device handler "[device handler filename]" reported a fatal error during initialization. Device may be missing or improperly configured. Skipping to next device. Press any key to continue.	Deallocate memory used to load driver; skip to next device in configuration file.
Device handler file does not contain device handler identification stamp: WARNING: Device handler "[device handler filename]" is not a valid DSI device handler. Skipping to next device. Press any key to continue.	Deallocate memory used for device handler; skip to next device in configuration file.
Device not available: No error message is displayed by the R&C program.	Insert error code 255 into return status field of DSRP; return to application software.

9. References

Thomason, B. L., Van de Wetering, B. L., & Booth, R. G. (March 1990) *A Portable Courseware Architecture (TN-90-11).* San Diego: Navy Personnel Research and Development Center.

Thomason, B. L., & Van de Wetering, B. L. (April 1990) *The MS-DOS Device Services Interface (TN-90-15).* San Diego: Navy Personnel Research and Development Center.

Van de Wetering B. L., & Thomason, B. L. (April 1990) *The MS-DOS Device Handler Interface (TN-90-16).* San Diego: Navy Personnel Research and Development Center.

Distribution List

Distribution:
Assistant Secretary of Defense (Force Management and Personnel)
Deputy Under Secretary of Defense for Research and Engineering (Research and Advanced Technology)
Director, Total Force Training and Education (OP-11)
Director, Aviation Training Systems Program Coordinator (PMA-205)
Commanding Officer, Naval Training Systems Center
Defense Technical Information Center (DTIC) (2)

Copy to:
Deputy Chief of Naval Operations (MP&T) (OP-01)
Assistant for Planning and Technical Development (OP-01B2)
Head, Training and Education Assessment (OP-11B1)
Director, Total Force Information System Management (OP-16)
Director, Submarine Manpower and Training Requirements (OP-29)
Director, Command Surface Warfare Manpower and Training Requirements (OP-39)
Director, Air ASW Training (OP594)
Assistant for Manpower, Personnel, and Training (OP-983D)
Commander, Naval Sea Systems Command (PMS 350)
Commander, Naval Sea Systems Command (PMS 396)
Commander, Naval Sea Systems Command (CEL-MP)
Director, Strategic Systems Project (SP-15)
Commanding Officer, New London Laboratory, Naval Underwater Systems Center (Code 33A)
Commanding Officer, New London Laboratory, Naval Underwater Systems Center (Code 3333)
Naval Training Systems Center, Technical Library (5)
Naval Training Systems Center (Code 10), (Code N-1), (Code 7)
Director, Office of Naval Research (OCNR-10)
Chief Scientist, Office of Naval Technology (OCNR-20T)
Chief of Naval Education and Training (Code 00)
Director, Training Technology (Code N-54)
Commanding Officer, Naval Education and Training Program Management Support Activity (Code 03) (2)
Commanding Officer, Naval Education and Training Program Management Support Activity (Code 04) (2)
Chief of Naval Technical Training (Code 00) (2)
Commander, Naval Military Personnel Command (NMPC-00/PERS-1)
Naval Military Personnel Command, Library (Code NMPC-013D)
Commanding Officer, Naval Health Sciences Education and Training Command, Bethesda, MD
Commander, Naval Reserve Force, New Orleans, LA
Commandant of the Marine Corps, Commanding General, Marine Corps Research and Development and Acquisition Command
Commander, U.S. ARI, Behavioral and Social Sciences, Alexandria, VA (PERTI-POT-I)
Technical Director, U.S. ARI, Behavioral and Social Sciences, Alexandria, VA (PERI-ZT)

Commander, Air Force Human Resources Laboratory, Brooks Air Force Base, TX
Scientific and Technical Information (STINFO) Office
TSRL/Technical Library (FL 2870)
Program Manager, Life Sciences Directorate, Bolling Air Force Base, DC (AFOSR/NL)
Commander, OPSTNGDIV Air Force Human Resources Laboratory, Williams Air Force Base, AZ (AFHRL/OT)
Commander, Air Force Human Resources Laboratory, Wright-Patterson Air Force Base, OH, Logistics and Human Factors Division (AFHRL/LRS-TDC)
Director of Training, Office of Civilian Personnel Management
Superintendent, Naval Postgraduate School
Director of Research, U.S. Naval Academy
Center for Naval Analyses